Generalization of Temporal Logic Tasks via Future Dependent Options

Philip S. Apple
papple@cs.umass.edu
College of Information and Computer Sciences
University of Massachusetts

Glen Banana
glen.banana@mila.quebec
Mila, Université de Montréal
CIFAR Canada AI Chair

Abstract

Temporal logic (TL) tasks consist of complex and temporally extended subgoals and they are common for many real-world applications, such as service and navigation robots. However, it is often inefficient or even infeasible for reinforcement learning (RL) agents to solve multiple TL tasks, since rewards are sparse and non-Markovian in these tasks. A promising solution to this problem is to learn task-conditioned policies which can zero-shot generalize to new TL tasks without further training. However, influenced by some practical issues, such as myopic use of options and long task horizon, previous works suffer from sub-optimality (or even infeasibility) in the found solution and sample inefficiency in training. In order to tackle these issues, this paper proposes a new option framework to train a generalist agent for multiple TL tasks. It consists of option training and task execution parts. In option training, in order to improve optimality, we propose to learn options dependent on the future subgoals. Additionally, in order to tackle the issue of long horizon in the training of future-dependent options, we propose to learn a multi-step value function which can propagate the rewards of satisfying future subgoals. Finally, we propose a model-free MPC planner to zero-shot solve unseen tasks with learned options. The generalization capability of the trained agent is evaluated in three environments, showing its significant advantage over previous methods. The source code is available at https://drive.google.com/drive/folders/128jGb3HkPbXPj-FO6FQ7NW3LTFkqUUQy?usp=drive_link

1 Introduction

Reinforcement learning (RL) is a promising framework for developing truly general agents capable of acting autonomously in the real world, ranging from video games (Mnih et al., 2015; Badia et al., 2020) to robotics (Levine et al., 2016; Inala et al., 2021). Previous RL algorithms primarily focus on solving tasks with a single goal state. However, many real-world applications may require agents to satisfy temporally extended goals (e.g., eventually take the key and then reach the door). Tasks consisting of temporally extended subgoals are termed as temporal logic (TL) tasks (De Giacomo & Vardi, 2013; Toro Icarte et al., 2018). TL tasks have a wide range of applications in the real world, such as control system and robotics. For example, a service robot on the factory floor might have to fetch the a set of components but in different orders depending on the product being assembled, and it may need to avoid some unsafe situations.

Generalization to multiple TL tasks is a key requirement for deploying autonomous agents in many real-world domains (Taylor & Stone, 2009). It is important for RL agent to learn to perform zero-shot execution of different tasks by leveraging the generalization abilities of deep learning models. However, previous related works suffer from various deficiencies (Kuo et al., 2020; Araki et al., 2021; Vaezipoor et al., 2021; den Hengst et al., 2022; Liu et al., 2022), such as lack of optimality or learning efficiency. Some works (Araki et al., 2021; den Hengst et al., 2022; Liu et al., 2022) solve new TL tasks by leveraging the learned reusable skills or options, but produce sub-optimal or even infeasible solutions, especially when the direct use of options can produce myopic solutions. These methods train every option for reaching a specific subgoal independently and myopically without considering the future, as we illustrated in an example in Section 3, which may fail

to achieve the global optimality of the task completion. Further, (Kuo et al., 2020; Vaezipoor et al., 2021) propose to train policies conditioned on the task formula directly, where the agent needs a large amount of environment samples to learn to understand temporal operators and figure out the optimal path for satisfying the task. These approaches have poor sample efficiency in complex tasks or environments, since they do not utilize reusable skills when solving compositional tasks like TL tasks.

In this work, in order to tackle the above issues, we propose a hierarchical option framework to solve the TL tasks and generalize to new tasks in a zero-shot manner. The proposed framework consists of option training part and task execution part.

In the *option training* part, we train a generalizable agent which can achieve various subgoals with different future situations. We propose two innovations for this part: 1) we introduce future-dependent options which are trained to not only achieve the target subgoal, but also consider other subgoals to be achieved in the future. By preventing myopic behavior in achieving the target subgoal, the proposed options can approximate the global optimality of the task completion as much as possible. 2) When the options are dependent on future subgoals, learning option policies needs to consider the achievement of subgoals in many time steps ahead. In order to facilitate the information propagation in this case, we train a multi-step value function to predict the discounted return of satisfying a future subgoal sequence.

In the *task execution* part, the given TL task is solved by a hierarchical option framework, where the high level is for option selection and the low level is for option execution. In the high level, whenever the previously selected option is successfully finished, we propose to use a model-free option planner to find the optimal sequence of subgoals which not only satisfies the given TL task and but also has the largest expected return in the multi-step value function, and then selects the option for reaching the first subgoal conditioned on other subgoals in the subgoal sequence. This option planner works in a manner of model predictive control (MPC) (Garcia et al., 1989), but it does not need to learn a transition model and hence circumvents the compounding errors caused by the inaccuracy of the learned transition model during planning.

In experiments, we demonstrate the zero-shot generalization capability of the trained agent in three environments, including both discrete and continuous action spaces. All these environments are procedurally generated where the layout and task specification are randomly generated, so that none of tasks can be solved by simple tabular methods (Sutton & Barto, 2018). With comprehensive evaluations, we show that the proposed approach outperforms previous representative methods in terms of sample efficiency, accuracy and optimality.

2 Preliminaries

2.1 Temporal Logic Task Specification

A temporal logic (TL) task is described by a TL specification φ, a Boolean function that determines whether the objective formula is satisfied by the given trajectory or not (Pnueli, 1977). The specification of TL task is used to express (multi-task) temporally extended subgoals and partial orders of subgoals for task completion (Huth & Ryan, 2004). First define a common vocabulary $\mathcal{AT}$ as the set of atomic tasks. TL tasks are widely used in real-world applications. For instance, in service robot applications, $\mathcal{AT}$ could include events such as opening the drawer, activating the fan, turning on/off the stove, or entering the bathroom. Then, the TL task can include temporally-extended occurrences of these events. For example, two possible TL tasks are (1) "Open the drawer and activate the fan in any order, then turn on the stove" and (2) "Open the drawer but do not enter the bathroom until the stove is turned off".

In order to study the systematic generalization of TL tasks with options, we adopt task temporal logic (TTL) (León et al., 2021) to specify TL tasks. TTL is designed to be an expressive, learning-oriented TL language interpreted over finite traces, i.e, over finite episodes. The language of TTL is a fragment of the widely-used Linear-time Temporal Logic over finite traces (LTLf) (De Giacomo & Vardi, 2013), and the translation of any TTL formula into LTLf is provably guaranteed in (León et al., 2020). TTL is expressive enough to represent tasks in (Andreas et al., 2017) which is a popular benchmark in the RL-TL literature.

Definition 1 (Task Temporal Logic (León et al., 2021)). Given the vocabulary $\mathcal{AT}$, every formula φ in TTL is built from atomic tasks $a \in \mathcal{AT}$, negation $\neg$ (on proposition only), and sequential composition ";", connected

by operators $\vee$, $\cup$, and $\mathcal{U}$. The grammar of TTL is expressed as below:

$$l ::= a|\neg a|l \vee l', \qquad \alpha ::= l\mathcal{U}l', \qquad \varphi ::= \alpha|\varphi;\varphi'|\varphi \cup \varphi' \tag{1}$$

where | indicates the alternative choices between templates. Specifically, an atomic task a represents the reachability of a corresponding subgoal in the environment. So, every atomic task a in TTL means that its associated subgoal a needs to be achieved eventually, i.e., $\Diamond a$ in LTL language (De Giacomo & Vardi, 2013). The operators $\vee$ and $\cup$ represent the non-deterministic choice. The operator $\mathcal{U}$ refers to "until" and $l\mathcal{U}l'$ reads l *must hold before* l' *is satisfied*. Besides, the sequential composition $\varphi;\varphi'$ represents that the formula φ' has to become satisfied after φ holds true. The temporal operators *eventually* $\Diamond$ and *always* $\Box$ can be also defined by operator $\mathcal{U}$.

Definition 2 (Satisfaction). The truth value of a TL task specification is determined by a finite sequence of truth assignments $\sigma = \langle \sigma_0, \sigma_1, \sigma_2, \ldots, \sigma_N \rangle$ with vocabulary $\mathcal{AT}$, where $a \in \sigma_i$ iff the atomic task a is achieved at time step i. Then, σ satisfies φ (1) at time $i \geq 0$, denoted by $\langle \sigma, i \rangle \models \varphi$:

- $\langle \sigma, i \rangle \models a$ iff $a \in \sigma_i$, where $a \in \mathcal{AT}$
- $\langle \sigma, i \rangle \models \neg a$ iff $\langle \sigma, i \rangle \not\models a \in \mathcal{AT}$
- $\langle \sigma, i \rangle \models (a \vee a')$ iff $\langle \sigma, i \rangle \models a$ or $\langle \sigma, i \rangle \models a'$
- $\langle \sigma, i \rangle \models l\mathcal{U}l'$ iff there exists j such that $i \leq j \leq N$ and $\langle \sigma, j \rangle \models l$, and $\langle \sigma, k \rangle \models l'$ for all $k \in [i, j)$
- $\langle \sigma, i \rangle \models \varphi;\varphi'$ iff there exists j such that $i < j \leq N$, and $\langle \sigma, i \rangle \models \varphi$, and $\langle \sigma, j \rangle \models \varphi'$
- $\langle \sigma, i \rangle \models (\varphi \cup \varphi')$ iff $\langle \sigma, i \rangle \models \varphi$ or $\langle \sigma, i \rangle \models \varphi'$

A sequence σ is defined to satisfy φ iff $\langle \sigma, 0 \rangle \models \varphi$. This sequence σ here is same as the sequence of symbolic observations in the rest of this paper.

Progression Technique. The progression function, denoted as $\text{prog}(\sigma, \varphi)$, is defined as a function which takes a TL specification and the current labelled state (symbolic observation) as inputs and returns a formula that expresses aspects of the original formula that remain to be satisfied (Bacchus & Kabanza, 2000; Vaezipoor et al., 2021). For example, in Figure 1, consider task $\varphi :=$ wood; diamond; ax, i.e., collect wood, then diamond and ax finally. It will be progressed to "diamond; ax" after wood is collected, meaning that the agent still needs to collect diamond and then ax.

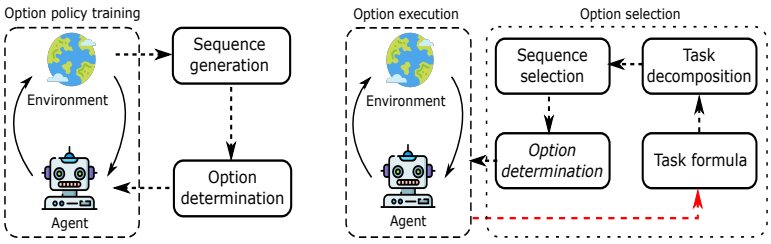
3 Methodology

In this work, we propose a novel future-dependent option framework to solve and generalize tasks with TL specifications. Here every option is trained to achieve a specific subgoal conditioned on future ones.

In the following, we first present a motivating example. Then the general framework of option training and task execution is illustrated. Finally, the algorithms, especially the model-free option planner, are introduced. Due to the space limit, the details of future-dependent option and multi-step value function are introduced in Appendix A.1 and A.2, respectively.

3.1 Motivation

In Figure 1, assume that environment reward R_e is -0.1 for every movement and the given task is $\varphi :=$ wood; diamond; ax (go to collect wood, then diamond and finally ax). There are two choices (s_A and s_B) for the agent to collect wood. Previous option-based approaches may myopically choose to collect the wood in the orange circle since it is closer, and finish the task φ along the green path. However, considering R_e, the globally optimal solution of task φ is the red path. In some cases, the decision made by myopic option-based approaches may lead to infeasible solutions. For instance, if the game in Figure 1 has constraint that the agent cannot move more than 12 steps in one episode, the green path with myopic choice of collecting wood is infeasible.



(a) Option Training (b) Task Execution

Figure 2: The proposed framework of option training and task execution. In task execution, the option selection part is a *model-free option planner*. The red arrow indicates that the task formula is progressed by the subgoal symbol g when g is achieved.

Therefore, although option framework was used an efficient approach to solve compositional RL problem like TL tasks in previous works (Kuo et al., 2020; Araki et al., 2021; Vaezipoor et al., 2021; den Hengst et al., 2022; Liu et al., 2022; León et al., 2021), myopically training each option to reach a single subgoal without considering the future can produce sub-optimal solutions.

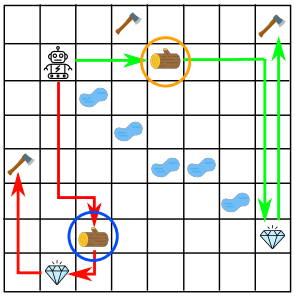


Figure 1: Motivating example. The TL task is wood; diamond; ax (go to collect wood, then diamond and finally ax). The state s_A denotes the state when the agent is at the wood in the orange circle, and s_B denotes that the agent is in the blue circle.

In this work, we propose a novel option framework where every option is trained with dependence on a sequence of future subgoals. We denote $\mathcal{O}$ as the set of options. Let o_g^ξ denote the option of reaching subgoal $g \in \mathcal{G}$ conditioned on ξ as a sequence of future subgoals to satisfy. We train each option $o_g^\xi \in \mathcal{O}$ not only by the experience of reaching the subgoal g, but also with the information of satisfying subgoals in ξ (in a fixed order same as ξ), since the option o_g^ξ is to reach subgoal g but also conditioned reaching the subgoal sequence ξ. Therefore, in order to back-propagate the information of satisfying subgoal sequence ξ (which often has long time horizon to complete), we also train a multi-step value function $V^\phi(s;\xi)$ to predict the discounted return obtained by reaching subgoals in ξ starting from the state s. We use V^ϕ to set target values to update value functions of options, hence accelerating the training efficiency of options. Consider the example in Fig. 1 again, when the option of collecting "wood" is also trained with reward information of collecting diamond and then ax after collecting wood, i.e., $g =$"wood" and $\xi :=$["diamond", "ax"], the option policy π_g^ξ will lead the agent to s_B (the wood in blue circle) instead of s_A. This is because $V^\phi(s_A;\xi) < V^\phi(s_B;\xi)$ and V^ϕ sets the targets for updating value function of option policy π_g^ξ.

3.2 General Framework

The proposed framework is presented in Figure 2. The option training part is to learn the policies of future-dependent options determined by subgoal sequences with various compositions and lengths. These learned options lay the foundation of agent's capability of generalization to new TL tasks. Every training iteration works as below. The details of training algorithm are presented in Algorithm 1 in Appendix A.11.

1. *Sequence generation:* the environment randomly generates subgoal sequences τ which only consist of subgoals from $\mathcal{G}$. For example, with $\mathcal{G} = \{a, b, c, d, e\}$, the subgoal sequence $\tau = [a, b, d]$ asks the agent to first reach a, then b and finally d, where a, b, d are subgoals;
2. *Option determination:* given the subgoal sequence τ, the agent needs to determine appropriate future dependent options to achieve subgoals in τ one-by-one. For example, if $\tau = [a, b, d]$, the options determined to be trained should be o_a^{bd}, o_b^d and $o_d^\varnothing$ ($\varnothing$ means empty);
3. *Option policy training:* the policies of determined options are trained by using an appropriate RL algorithm together with agent's experience of trying to satisfy subgoals in τ. During the training, the

reward propagation is augmented by the multi-step value function $V^{\phi}(\cdot;\tau)$ which is introduced in Section A.2;

The task execution part in Figure 2(b) is a hierarchical framework for zero-shot solving new TL tasks by using learned options, where the option selection is conducted in the high level and the policy of selected option is executed in the low level. In the high level, the option selection is realized by a *model-free option planner*, where the task formula is first decomposed into the set of satisfying subgoal sequences, then the optimal subgoal sequence is selected and finally the optimal option to be executed is determined. In the low level, the policy of selected option is executed in the environment. More details about option planner are introduced in Section 3.3.1.

1. *Task decomposition:* The given task φ is first decomposed into a set $\mathcal{K}$ of subgoal sequences τ_i, i.e., $\mathcal{K} := \{\tau_i\}_{i=1}^{M_\varphi} = \{[g_1^i, g_2^i, \ldots, g_{L_i}^i]\}_{i=1}^{M_\varphi}$, where each τ_i can satisfy φ and any g_j^i is from $\mathcal{G}$. For instance, if $\varphi = a; (b \vee c); (d \vee e)$, the decomposed subgoal sequences are $\mathcal{K} = \{[a,b,d],[a,b,e],[a,c,d],[a,c,e]\}$ and $M_\varphi = 4$. The details are in Algorithm 3 in Appendix;

2. *Sequence selection:* The optimal subgoal sequence τ_{i^*} with highest expected return is selected from $\mathcal{K}$ according to the multi-step value function V^{ϕ}, i.e., $i^* = \arg\max_{i\in[1,M_\varphi]} V^{\phi}(s_0;\tau_i)$ and s_0 is the initial state;

3. *Option determination:* Different from that in the option training part, only the *first* option from τ_{i^*}, i.e., $o_{\tau_{i^*}[0]}^{\tau_{i^*}[1:]}$, is determined and sent to the low level for execution. For example, assuming $\tau_{i^*} = [a,b,d]$, only o_a^{bd} is selected and executed in the low level;

4. *Option policy execution:* The agent executes the policy of option selected above in the environment. When the selected option is successfully finished, the task formula φ is updated by the progression with the achieved subgoal. For example, assuming that the target task is $\varphi = a; (b \vee c); (d \vee e)$ and the execution of option o_a^{bd} is successfully finished, the task formula φ will be progressed by the subgoal symbol a and become $\varphi := \text{prog}(\varphi, a) = (b \vee c); (d \vee e)$.

Remark. The future-dependent property makes the option selection non-Markovian, and this is reason for performance improvement. In previous option frameworks, such as (León et al., 2020; 2021), every option is trained to reach a specific subgoal, and option selection is independent of other options and hence Markovian. However, this is shown to be sub-optimal in the motivating example of Section 3.1, where making options dependent on the future can produce globally optimal solution. But future-dependent option loses the Markovian property, since its policy is dependent on future subgoals. *The details of future-dependent option and multi-step value function are introduced in Section A.1 and A.2 in Appendix.*

3.3 Algorithms

The algorithms for training and testing are in Algorithm 1 and 2 in Appendix A.11, and the algorithm for subgoal extraction is presented in Algorithm 3. The task generation for performance evaluation is introduced in Appendix A.3. More practical implementation techniques are in Appendix A.10.

3.3.1 Model-free Option Planner

In the high level of TL task execution, we use a model-free planner to determine the next option to execute. As shown in Figure 2(b), it uses the standard model-predictive control (MPC) technique (Garcia et al., 1989): first extracts all the subgoal sequences satisfying the target task ϕ, then finds the optimal subgoal sequence ξ^* according to multi-step value function $V^{\phi}(s;\xi)$ and finally determines the option of achieving first subgoal $\xi^*[0]$ conditioned on achieving future ones $\xi^*[1:]$, i.e., $o_{\xi^*[0]}^{\xi^*[1:]}$, to execute. Different from previous MPC planners, we do not need to learn a transition model here since V^{ϕ} is already trained to predict the expected return of achieving various subgoal sequences ξ. So, although previous model-based planners suffer from compounding errors (Asadi et al., 2018; Lambert et al., 2022), this issue can be avoided by our proposed

(a) Letter (b) Room (c) Navigation

Figure 3: Environments. Note that these environments are procedurally generated and hence tasks cannot be solved by simple tabular methods.

model-free planner. Whenever the selected option is successfully finished, the task formula needs to be updated by progression with the symbol of achieved subgoal, as indicated by the red arrow in Figure 2(b).

Although function V^ϕ is trained to predict the return of finite subgoal sequences which have maximum length K, V^ϕ is still generalizable to longer sequences, since the representation of subgoal sequence is extracted by GNN which has strong power of generalization. This is validated by the following experiments.

During the task execution, in order to avoid unsafe symbols, whenever task formula is updated the agent finds a set of unsafe symbols $\mathcal{U}_{\text{unsafe}}$ which can falsify the current task φ, i.e., $\mathcal{U}_{\text{unsafe}} = \{q | q \in \mathcal{G}, \text{prog}(q, \varphi) = \text{false}\}$. In the low level of the task execution framework, actions a_t which can lead the agent too close to symbols in $\mathcal{U}_{\text{unsafe}}$ will be ignored, where the closeness is measured by the action value function $Q_g^\varnothing$ for $\forall g \in \mathcal{U}_{\text{unsafe}}$. The details of task execution are presented in Algorithm 2 in Appendix.

4 Experiments

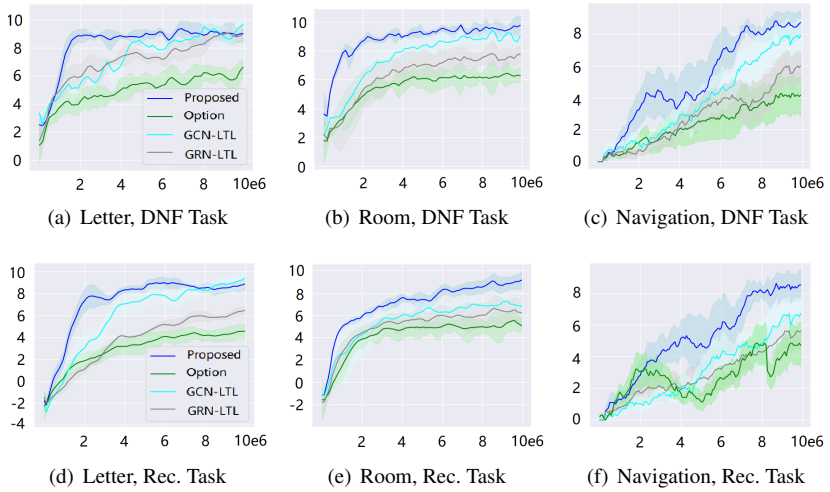
Our experiments are designed to evaluate the performance of multi-task RL agent trained by the proposed framework, including sample efficiency, optimality and generalization. Specifically, we focus on the following questions: 1) **Performance**: whether the proposed framework can outperform previous representative methods in terms of optimality and sample efficiency; 2) **Ablation study**: what is the influence of different components of the proposed framework on the learning performance; 3) **Long horizon tasks**: whether the proposed framework can train the multi-task agent to better solve long-horizon unseen tasks; 4) **Visualization**: what the learned value function looks like for options conditioned on different future subgoals. The baselines for performance comparison are presented in Appendix A.4. The neural architecture and hyper-parameters used in experiments are also introduced in Appendix A.9 and A.12.

4.1 Environments

We conducted experiments across different environments and TL tasks, where the tasks vary in length and difficulty. All the environments are procedurally generated, where the layout and positions of objects are randomly generated upon reset. The positions and properties of objects are unknown to the agent. As such, none of the environments adopted here can be solved by simple tabular-based methods.

Letter. This environment is a $n \times n$ grid game which is a variant of that in Figure 1, replacing objects by letters. Out of the n^2 grid cells, m grids are associated with k (where $m > k$) unique propositions (letters). Note that some letters may appear in multiple cells, giving the option of reaching them in multiple ways. An example layout is shown in Figure 3(a) with $n = 7, m = 10$ and $k = 5$. At each step the agent can move along the cardinal directions (up, down, left and right). The agent is given the task specification and is assumed to observe the full grid (and letters) from an egocentric point of view with the image-based observation. Each task is described by a TL formula in terms of these letters. But positions of these letters are unknown to the agent. The agent must visit these letters' locations in certain way to satisfy the TL formula.

Room. This environment is also a grid-world game, but its observation is divided into four rooms as shown in Figure 3(b). There are 5 letters located in 8 positions, corresponding to 5 propositions randomly allocated in these rooms. An example of layout is shown in Figure 3(b). The agent is randomly placed into one of these



(a) Letter, DNF Task (b) Room, DNF Task (c) Navigation, DNF Task

(d) Letter, Rec. Task (e) Room, Rec. Task (f) Navigation, Rec. Task

Figure 4: Performance Comparisons. The x-axis is the environmental step, and the y-axis is the return. "Rec." is short for recursive. The first row is for evaluating DNF tasks, and the second row is for evaluating recursive tasks. The definitions of DNF and recursive tasks are in Appendix A.3. The return is defined as the sum of rewards along the trajectory.

rooms. Each room is connected to its neighbors by corridors. Two randomly selected corridors are blocked by locks. The agent can open a lock by using a key corresponding to that specific lock (having the same color). These (green and yellow) keys are placed in positions which the agent can reach. This environment is an upgrade of MineCraft with obstacles and dependencies between objects imposed by keys and locks. The observation is also image-based here and the agent does not know the positions of objects. Every task formula is a TL formula in terms of object's letters. The agent must visit these letters' locations in certain way to satisfy the TL formula.

Navigation. This is a robotic environment with continuous action and state spaces. It is modified from OpenAI's Safety Gym (Ray et al., 2019). As shown in Figure 3(c), the environment is a 2D plane with 6 to 9 colored circles, called "navigation". Here each color represents a proposition in task specification, with some circles sharing the same color. We use Safety Gym's Point robot whose actions are steering and forward/backward acceleration. Its observation includes the lidar information towards the circles and other sensory data (e.g., accelerometer, velocimeter). The circles and the robot are randomly positioned on the plane at the start of each episode and the robot has to visit and/or avoid certain colors in a particular manner described by the TL specification.

4.2 Training and Evaluation

In every training episode, the agent uses appropriate option policies to satisfy a subgoal sequence ξ which is randomly selected from the set of subgoals of the environment. After every fixed number of training steps or episodes, the agent is evaluated on a fixed number of tasks with TL specifications randomly sampled from a large set of possible tasks (more than 10^6). The task generation for evaluation is introduced in Section A.3 in Appendix. In Appendix A.6, we also evaluate the agent on TL tasks whose solution has longer horizons than subgoal sequences used in the training stage, verifying the generalization of the trained agent to more difficult tasks. The details of option training and task evaluation algorithms are presented in Appendix A.11.

Remark. The algorithm performance reported here can be also regarded as the evaluation of zero-shot generalization. First, since the agent is only trained to complete subgoal sequences, the TL tasks in evaluation are unseen in the training. Second, the agent directly applies the policies of trained options to complete tasks in evaluation, so that no further learning is needed for any unseen tasks in the evaluation. These two arguments also hold in baselines (see Appendix A.4). Therefore, the zero-shot generalization capability of the proposed framework is evaluated and compared with baselines.

4.3 Results

In this section, we present the comparison results of the proposed method with baselines. The overall performance comparisons in terms of average return for satisfying TL tasks are first presented. Then, we demonstrate the ablation studies to investigate the effects of different components of the proposed framework. More ablation study on the multi-step value function in the navigation domain is shown in Section A.5. In each plot of the proposed framework, the x-axis is the number of environment steps used in the option training algorithm, while the y-axis is the evaluation performance of task execution algorithm by applying trained options. In evaluation, the task is randomly generated according to some template.

The ablation study, performance in long-horizon task and visualization of value function are presented in Appendix A.5, A.6 and A.7, respectively.

4.3.1 Performance

In Figure 4, the proposed method is compared with three baselines introduced in Section A.4. We can see that although Baseline-1 can learn fast in the early stage, its overall performance is the worst. The optimality in Baseline-1 degrades because the resulting options myopically focus on the next subgoal only, without looking ahead. It shows the importance of the dependence of options on future subgoals. In addition, the proposed method can learn much faster than GCN-LTL and GRU-LTL, confirming that leveraging reusable skills via options can achieve better sample efficiency. Further, the agents in GCN-LTL and GRU-LTL, which are conditioned on the task specification directly, need a lot of environment samples to understand temporal operators and find out the optimal path in the formula to finish the task.

5 Conclusion

In this work, we propose a novel framework for generalizing TL tasks by options dependent on the future subgoal sequence. Moreover, to facilitate the reward propagation of satisfying future subgoals, we propose to learn a multi-step value function updated by Monte Carlo estimates of discounted return. Based on these, we also propose a new model-free option planner for task execution, which circumvents the compounding errors caused by the learned transition model. With comprehensive experiments, the proposed method is confirmed to have significant advantages over previous methods in terms of optimality and sample efficiency.

The limitation of this proposed framework is that it cannot solve TL tasks with noisy symbolic observations (Li et al., 2022). In real-world applications the values of symbols or propositions can be stochastic, and the achievement of subgoals and even tasks can be probabilistic. In the future work, we will extend the proposed framework to handle TL task with noisy symbolic observations by learning a probabilistic model of symbols conditioned on the input observation.

References

Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *International Conference on Machine Learning*, pp. 166–175. PMLR, 2017.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

Brandon Araki, Xiao Li, Kiran Vodrahalli, Jonathan DeCastro, Micah Fry, and Daniela Rus. The logical options framework. In *International Conference on Machine Learning*, pp. 307–317. PMLR, 2021.

Kavosh Asadi, Dipendra Misra, and Michael Littman. Lipschitz continuity in model-based reinforcement learning. In *International Conference on Machine Learning*, pp. 264–273. PMLR, 2018.

Fahiem Bacchus and Froduald Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial intelligence*, 116(1-2):123–191, 2000.

Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. Agent57: Outperforming the atari human benchmark. In *International Conference on Machine Learning*, pp. 507–517. PMLR, 2020.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Giuseppe De Giacomo and Moshe Y Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI'13 Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pp. 854–860. Association for Computing Machinery, 2013.

Floris den Hengst, Vincent François-Lavet, Mark Hoogendoorn, and Frank van Harmelen. Reinforcement learning with option machines. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pp. 2909–2915. International Joint Conferences on Artificial Intelligence Organization, 2022.

Carlos E Garcia, David M Prett, and Manfred Morari. Model predictive control: Theory and practice—a survey. *Automatica*, 25(3):335–348, 1989.

Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge university press, 2004.

Jeevana Priya Inala, Yecheng Jason Ma, Osbert Bastani, Xin Zhang, and Armando Solar-Lezama. Safe human-interactive control via shielding. *arXiv preprint arXiv:2110.05440*, 2021.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Yen-Ling Kuo, Boris Katz, and Andrei Barbu. Encoding formulas as deep networks: Reinforcement learning for zero-shot execution of ltl formulas. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5604–5610. IEEE, 2020.

Nathan Lambert, Kristofer Pister, and Roberto Calandra. Investigating compounding prediction errors in learned dynamics models. *arXiv preprint arXiv:2203.09637*, 2022.

Borja G León, Murray Shanahan, and Francesco Belardinelli. Systematic generalisation through task temporal logic and deep reinforcement learning. *arXiv preprint arXiv:2006.08767*, 2020.

Borja G León, Murray Shanahan, and Francesco Belardinelli. In a nutshell, the human asked for this: Latent goals for following temporal specifications. In *International Conference on Learning Representations*, 2021.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Andrew C Li, Zizhao Chen, Pashootan Vaezipoor, Toryn Q Klassen, Rodrigo Toro Icarte, and Sheila A McIlraith. Noisy symbolic abstractions for deep rl: A case study with reward machines. In *Deep Reinforcement Learning Workshop NeurIPS 2022*, 2022.

Jason Xinyu Liu, Ankit Shah, Eric Rosen, George Konidaris, and Stefanie Tellex. Skill transfer for temporally-extended task specifications. *arXiv preprint arXiv:2206.05096*, 2022.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pp. 46–57. ieee, 1977.

Alex Ray, Joshua Achiam, and Dario Amodei. Benchmarking safe exploration in deep reinforcement learning. *arXiv preprint arXiv:1910.01708*, 7:1, 2019.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pp. 593–607. Springer, 2018.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Sungryull Sohn, Junhyuk Oh, and Honglak Lee. Hierarchical reinforcement learning for zero-shot generalization with subtask dependencies. *Advances in Neural Information Processing Systems*, 31, 2018.

Shao-Hua Sun, Te-Lin Wu, and Joseph J Lim. Program guided agent. In *International Conference on Learning Representations*, 2019.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009.

Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. Teaching multiple tasks to an rl agent using ltl. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 452–461, 2018.

Pashootan Vaezipoor, Andrew C Li, Rodrigo A Toro Icarte, and Sheila A Mcilraith. Ltl2action: Generalizing ltl instructions for multi-task rl. In *International Conference on Machine Learning*, pp. 10497–10508. PMLR, 2021.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

A Appendix

A.1 Future Dependent Option

In this work, we define a future dependent option, i.e., $o_g^\xi := \langle \mathcal{S}, \beta_g, \pi_g^\xi \rangle$ where ξ is a finite sequence of subgoals to be achieved in the future after completing g. Each option is trained to achieve its corresponding subgoal $g \in \mathcal{G}$ conditioned on future subgoals in ξ, working in the multi-task MDP. For each option, without loss of generality, the initial set is the same as the state space $\mathcal{S}$, and the terminal function is the indicator of satisfying the subgoal g, i.e., $\beta_g(s) = \mathbf{1}\{L(s) \models g\}$ where $L(\cdot)$ is the labeling function defined in Section 2.1. The option policy π_g^ξ is trained to maximize the discounted return of achieving the target subgoal g conditioned on achieving the future subgoals in sequence ξ, encouraging the option policy to realize the global optimality of achieving both g and ξ. Specifically, since achieving the future subgoal sequence ξ is taken into consideration, the return of achieving ξ needs to be back-propagated to train the action value (Q) function of the option π_p^ξ, which is helped by the multi-step value function V^ϕ introduced in next section.

Generally, when the option policy is trained by an off-policy method, the agent learns a sample-based approximation to the Q function $Q_{\pi_g^\xi}(s, a)$ of option o_g^ξ, denoted as $Q_g^\theta(s, a; \xi)$ referring to the expected discounted return of achieving subgoal g conditioned on the achievement of subgoal sequence ξ in the future. Alternatively, when the option policy is trained by an on-policy method, the agent learns a value function $V_g^\theta(s; \xi)$ to approximate the value function $V_{\pi_g^\xi}(s)$ of option o_g^ξ. The Q (or V) function of option is updated by TD-1 method (Sutton & Barto, 2018). For different environments, we choose an appropriate RL method to learn option policies, such as SAC for off-policy or PPO for on-policy method.

In option training, the option policy π_g^ξ is trained together with other options used to satisfy future subgoals in ξ. Specifically, given any subgoal sequence $\xi := \{g_i\}_{i=1}^K$ in option training, we define the sub-sequences $\xi_k := \{g_i\}_{i=k+1}^K$ and $k = 1, \ldots, K$. We then start from trying the option policy $\pi_{g_1}^{\xi_1}$ to achieve subgoal g_1. When the subgoal g_k in ξ is satisfied, we switch to use another option policy $\pi_{g_{k+1}}^{\xi_{k+1}}$ to achieve g_{k+1}, repeating this process until the agent satisfies the last subgoal g_K by using the option policy $\pi_{g_K}^{\varnothing}$. In addition to environmental rewards, the agent will receive the reward R_F when the last subgoal g_K is satisfied. For any $k = 1, \ldots, K-1$, the discounted returns during the executions of option policies from $\pi_{g_{k+1}}^{\xi_{k+1}}$ to $\pi_{g_K}^{\varnothing}$ are all back-propagated to train the policy $\pi_{g_k}^{\xi_k}$ (updating $Q_g^\theta(\cdot, \cdot; \xi_k)$ or $V_g^\theta(\cdot; \xi_k)$), via the multi-step value function V^ϕ introduced in Section A.2.

It is worth noting the difference between V_g^θ and V^ϕ. The value function $V_g^\theta(\cdot; \xi)$ is used to train option policy π_g^ξ if on-policy training method is used, and it is associated with the option o_g^ξ and subgoal g. However, $V^\phi(\cdot; \xi)$ is the multi-step value function used to propagate reward information of satisfying the subgoal sequence ξ, independent of any options or subgoals.

A.2 Multi-step Value Function

Since the value functions of option policies (Q_g^θ or V_g^θ) are updated with TD-1 method in (Sutton & Barto, 2018), each update can propagate the reward information for only one time step. However, we note that the satisfaction of a future subgoal sequence ξ can have long horizon with sparse rewards, and the training of option policy π_g^ξ is dependent on satisfying ξ. Therefore, it can be inefficient to propagate the reward information of satisfying ξ back to update Q_g^θ or V_g^θ in training the option policy π_g^ξ. In the rest of the paper, we use $\xi[k]$ to denote the k-th subgoal in sequence ξ.

In order to help the propagation of reward information in long-horizon tasks, we propose to learn a multi-step value function $V^\phi(s; \xi)$ to estimate the discounted return of satisfying the subgoal sequence ξ starting from state s. In option training, the output of V^ϕ is used to set the target value for updating $Q_g^\theta(\cdot, \cdot; \xi)$ or $V_g^\theta(\cdot; \xi)$ so that the reward propagation toward option policies can be accelerated. In Figure 5, it visually shows how the reward information is back-propagated in both Q^θ and V^ϕ when options are learned by an off-policy RL method. In the on-policy case, V^θ and V^ϕ work in the same way. Additionally, this function $V^\phi(\cdot; \xi)$ is also used to build a model-free option planner in task execution.

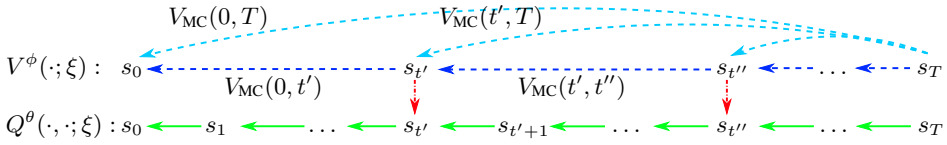


Figure 5: Diagram of back-propagation of reward information. The green line shows that Q functions of different options (target subgoals are omitted) are learned by TD-1 method. Note that function V^{ϕ} is independent of subgoals while Q^{θ} is dependent on different subgoals. The first and second subgoals ($\xi[0]$ and $\xi[1]$) are satisfied at $s_{t'}$ and $s_{t''}$, respectively. The red line shows that the multi-step value function V^{ϕ} sets the target value for Q^{θ} whenever a subgoal is satisfied. The blue and cyan curves denote the Monte Carlo estimate of multi-step discounted return $V_{\text{MC}}(\cdot,\cdot)$. V^{ϕ} is updated whenever a subgoal is satisfied, which has much coarser time resolution than Q^{θ}.

Specifically, the target value for updating V^{ϕ} is calculated based on Monte Carlo (MC) estimates of two discounted returns. 1) The first is the MC estimate of the discounted return till the end of the trajectory (cyan curves in Figure 5), i.e., $V_{\text{MC}}(t,T) := \sum_{k=t}^{T} \gamma^{k-t} r_t$ (T is the last time step of the trajectory). We use $V_{\text{MC}}(t,T)$ here since it is unbiased and good at capturing long-term rewards, but it also has large variance (Sutton & Barto, 2018); 2) In order to attenuate the variance, we also use the MC estimate of the discounted return till the satisfaction of next subgoal $\xi[0]$ (blue curves in Figure 5), i.e., $V_{\text{MC}}(t,t') = \sum_{k=t}^{t'} \gamma^{k-t} r_t$ (t' is the time when $\xi[0]$ is satisfied). This $V_{\text{MC}}(t,t')$ is used to build a multi-step temporal difference (TD) target for updating V^{ϕ}, which is together with the value estimate of satisfying other subgoals $\xi[1:]$ from a lagged value network V^{ϕ^-} (Van Hasselt et al., 2016).

Assume we have a trajectory $\tau = \{s_0, a_0, r_0, s_1, \ldots, s_{T-1}, a_{T-1}, r_{T-1}, s_T\}$ with the subgoal sequence ξ as the target task to complete. If next subgoal $\xi[0]$ is satisfied at time t', the target for multi-step V^{ϕ} function is written as

$$V^{\text{target}}(s_t;\xi) = \max\{V_{\text{MC}}(t,t') + \gamma^{t'-t} V^{\phi^-}(s_{t'};\xi[1:]), V_{\text{MC}}(t,T)\} \tag{2}$$

In the equation above, the first term in the maximum is a multi-step TD target formed by $V_{\text{MC}}(t,t')$ together with the lagged value network V^{ϕ^-}. As discussed above, we also use $V_{\text{MC}}(t,T)$ in (2) to directly capture the reward information till the end of the trajectory. Since the value network always has very small values throughout the state space in early training stages, we need to use a maximum operator in (2) to help the reward back-propagate from the end of the trajectory. If next subgoal $\xi[0]$ is not satisfied by any state in τ, the target will become $V^{\text{target}}(s_t;\xi) = \max\{V^{\phi^-}(s_t;\xi), V_{\text{MC}}(t,T)\}$. Finally, the value function V^{ϕ} is trained to predict its target value by minimizing the loss function

$$J(\phi) = \ell(V^{\phi}(s_t;\xi), V^{\text{target}}(s_t;\xi)) \tag{3}$$

where ℓ is an arbitrary differentiable loss function.

The value function V^{ϕ} also sets the target value to update the Q functions of option policies (i.e., $Q_g^{\theta}(\cdot,\cdot;\xi)$) whenever the subgoal g is satisfied. For any tuple (s_t, a_t, r_t, s_{t+1}), the target value for $Q_g^{\theta}(\cdot,\cdot;\xi)$ is expressed as,

$$\begin{aligned} Q_g^{\text{target}}(s_t, a_t, r_t, s_{t+1};\xi) &= r_t + \gamma\beta_g(s_{t+1}) V^{\phi}(s_{t+1};\xi) \\ &+\gamma(1-\beta_g(s_{t+1})) \max_{a'} Q_g^{\theta^-}(s_{t+1}, a';\xi) \end{aligned} \tag{4}$$

where θ^- is the parameter of the lagged target network as (Van Hasselt et al., 2016). This target means that when g is not satisfied yet (i.e., $\beta_g(s_{t+1})$ =false), the Q function is updated via the classical TD-1 method. However, whenever g is satisfied (i.e., $\beta_g(s_{t+1})$ =true), it is updated with the target value given by $V^{\phi}(\cdot;\xi)$ which can quickly propagate discounted return (reward information) of satisfying ξ back to s_{t+1}, achieving the global optimality of satisfying both g and ξ. Then $Q_g^{\theta}(\cdot,\cdot;\xi)$ can be updated by minimizing the loss as,

$$J(\theta) = \mathbb{E}_{(s,a,r,s',g,\xi)\sim\mathcal{B}}\left[\ell(Q_g^{\theta}(s,a;\xi), Q_g^{\text{target}}(s,a,r,s';\xi))\right] \tag{5}$$

where $\mathcal{B}$ is the replay buffer.

A.3 Task Generation

We define the *depth* of a task formula φ as the length of the shortest subgoal sequence to satisfy φ. We generate two kinds of tasks to evaluate agent's performance.

The first kind of task is the "DNF" task described by a disjunctive normal formula that concatenates terms by disjunctive operator $\cup$, i.e., $\varphi_{dnf} = \varphi_{dnf} \cup \varphi'$ and $\varphi' = \varphi'; s|\varphi'; \neg g$. Here, s and g are propositions denoting two different subgoals. The notation | denotes alternative. When generating a task formula, two sub-formulae around | are randomly selected. For instance, $\varphi_{\text{dnf}} = (a; b; \neg e) \cup (c; d)$. Specifically, the number of terms that are connected with the disjunctive operator ranges between 3 and 5, and the number of propositions in every term is between 1 to 5.

The second type of task is called "Recursive" task, which can be formulated as $\varphi_{\text{rec}} = \varphi_{\text{rec}}; \varphi'|\varphi_{\text{rec}} \cap \varphi'$ and $\varphi' = g \vee \varphi'|\neg s\mathcal{U}(g \vee \varphi')|\neg s\mathcal{U}g$. Here, s and g are propositions denoting two different subgoals. The depth of the recursion is randomly selected between 3 and 5. An example of "recursive" task is $(\neg a \cup (b \vee c)); e; (\neg f \cup g)$, and the shortest subgoal sequence for satisfying this task is $b \to e \to g$ or $c \to e \to g$ with the depth of 3.

A.4 Baselines

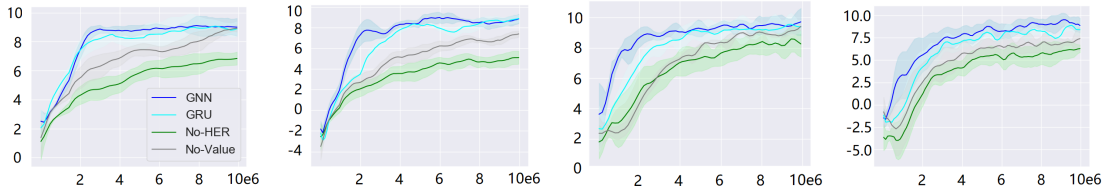
The proposed algorithm is compared with three baselines. The model architecture and hyper-parameters of the proposed method and baselines are introduced in Appendix A.9 and A.12. The first baseline (*Option*) is based on the conventional option framework, where a planning technique is used to tell the agent which proposition to achieve next and every option is only trained to achieve next proposition as the subgoal. This idea was widely used by previous works on multi-task RL (Andreas et al., 2017; Sohn et al., 2018; Sun et al., 2019; León et al., 2020; Araki et al., 2021; León et al., 2021). The agent's model here is the same as that in our method, except that the options are myopic and do not consider future subgoals. In order to make comparisons to be fair, in *Option* baseline the RL algorithms for training the agent and hyper-parameters are the same as the proposed method, where HER and formula transformation are both adopted. However, in *Option* baseline, since options are not conditioned on future subgoals, their training does not need future rewards and the multi-step value function V^ϕ is not used.

The second baseline (*GCN-LTL*) is modified from (Vaezipoor et al., 2021), where the task formula is processed by a graph convolutional network (GCN) (Kipf & Welling, 2016) and progresses over time. The architecture of GCN here is the same as that in (Vaezipoor et al., 2021) with $T = 8$ message passing steps and 32-dimensional node embedding. Other parts of agent's model are the same as the proposed method. The third baseline (*GRU-LTL*) is based on the method in (Kuo et al., 2020). This approach trains an agent that considers the whole task specification as an extra input and uses GRU (Chung et al., 2014) to learn an embedding of the TL specification which does not progress over time. The learned task embedding has the size of 32 which is same as the size of embedding of future subgoals in our method. Other parts of agent's model are the same as the proposed method.

In original papers of GCN-LTL and GRU-LTL (Kuo et al., 2020; Vaezipoor et al., 2021), the agent is trained by on-policy PPO algorithms. In order to make them comparable with the proposed framework, GCN-LTL and GRU-LTL use same RL algorithm as our framework, with the same hyperparameters as ours. In the letter and room domains, the agents in GCN-LTL and GRU-LTL are trained by the off-policy Q learning (Mnih et al., 2015) approach. In the navigation domain, GCN-LTL and GRU-LTL still use the PPO algorithm. Since the agent takes the original TL specification as its input directly, formula transformation and HER cannot be used in GCN-LTL or GRU-LTL. Since their original implementations are not option-based, the multi-step value function V^ϕ is not used either.

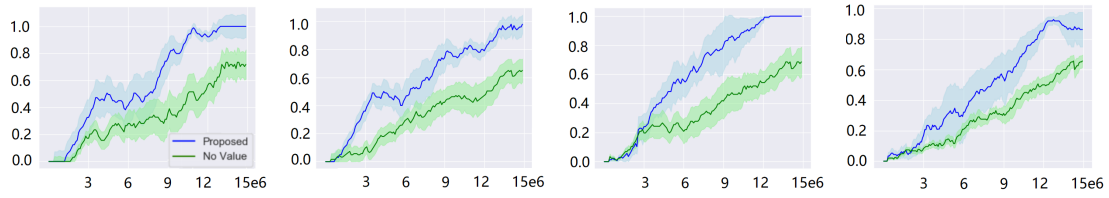
A.5 Ablation Study

The ablation study is first to examine the difference between GNN and GRU when used in option critics $Q_p^\theta(\cdot, \cdot; \xi)$ and value function $V^\phi(\cdot; \xi)$ to learn the embedding of the sequence ξ of future subgoals. Specifically, the nodes of GNN represent subgoals and every subgoal is connected to its successor by a directed edge. The embedding of sequence ξ is learned by GCN with multi-step message passing ($T = 8$). In addition, when the



(a) Letter, DNF Task (b) Letter, Rec. Task (c) Room, DNF Task (d) Room, Rec. Task

Figure 6: Ablation study. The x-axis is the environmental step, and the y-axis is the return. "No-HER" refers to the proposed method without using HER. "No-value" refers to the proposed method without using the multi-step value function.



(a) Nav1, DNF Task (b) Nav2, DNF Task (c) Nav1, Rec. Task (d) Nav2, Rec. Task

Figure 7: The ablation study of the multi-step value function V^ϕ in navigation domain. The x-axis is the environmental step, and the y-axis is the success rate.

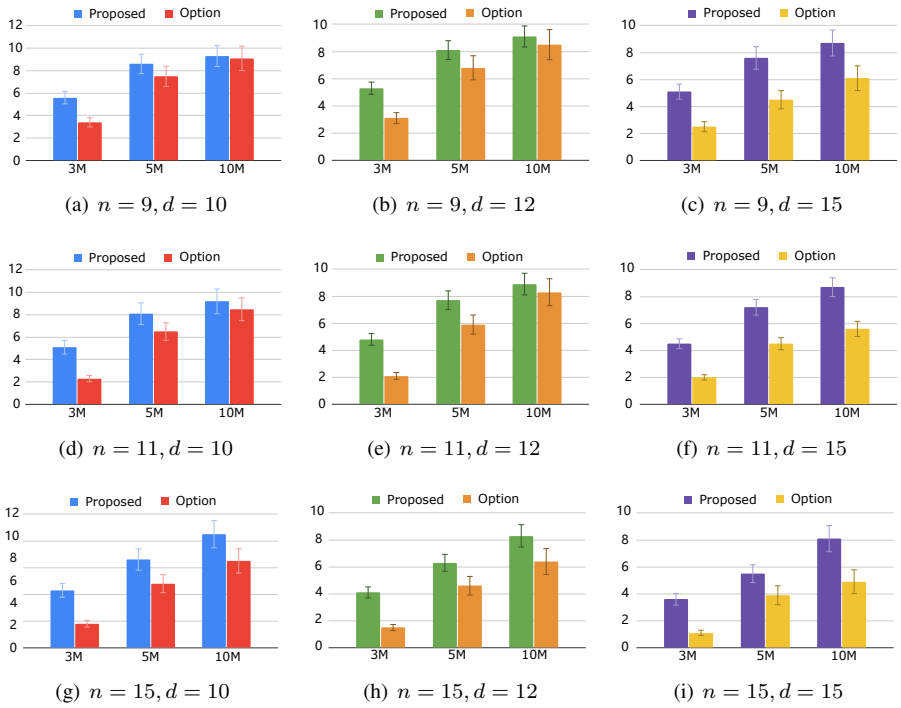
GRU is used, sequence ξ, with every element one-hot encoded, is fed into GRU and the embedding can be obtained at the output of GRU. More details of agent's model are in Appendix. In Figure 6, we can see that the GRU performs slightly worse than GNN.

In addition, we study the effects of multi-step value function V^ϕ and HER by comparing "No-value" and "No-HER" with the proposed method in Figure 6. We can see that when V^ϕ or HER is not used, the learning performance can degrade significantly, implying their importance in performance improvement. Since the time horizon of tasks in the navigation domain is much longer than other domains, we find that the multi-step value function V^ϕ plays a more important role in navigation domain. The reward propagation is more difficult when task's time horizon increases, and hence the usage of V^ϕ can improve the sample efficiency of our framework significantly in this case.

Multi-step Value Function. In navigation domain, the time horizon of every task is 1000 which is much longer than that in other domains. The experiments in this section are conducted in two environments of navigation domain. The first environment has 3 colors and 6 objects, denoted as "Nav1" whereas the second environment has 5 colors and 10 objects, denoted as "Nav2". Both DNF and Recursive tasks are evaluated in these two environments. The results of ablation study on the multi-step value function V^ϕ are shown in Figure 7, where the curve of "No Value" refers to our framework without using V^ϕ. Compared with other experiment results, we can see that V^ϕ can improve the sample efficiency more in navigation domain.

A.6 Long Horizon Tasks

In order to verify the effectiveness of the reward propagation, we evaluate the performance of the trained RL agent in tasks with long time horizon. We focus on the letter domain where the map size and the depths of the TL specification are changed for comparison. The depth of a formula φ is the length of optimal subgoal sequence to satisfy φ. Baseline Option only learns independent option for each subgoal and does not consider reward propagation. Baseline GRU-LTL uses recurrent GNN to process the TL specification not progressed, so its performance on TL tasks with long horizon is much worse than Baseline GCN-LTL. Therefore, we do not consider Baseline-1 and Baseline GRU-LTL for comparison here. In every experiment, there are 8 unique letters on the map and every letter appears twice.

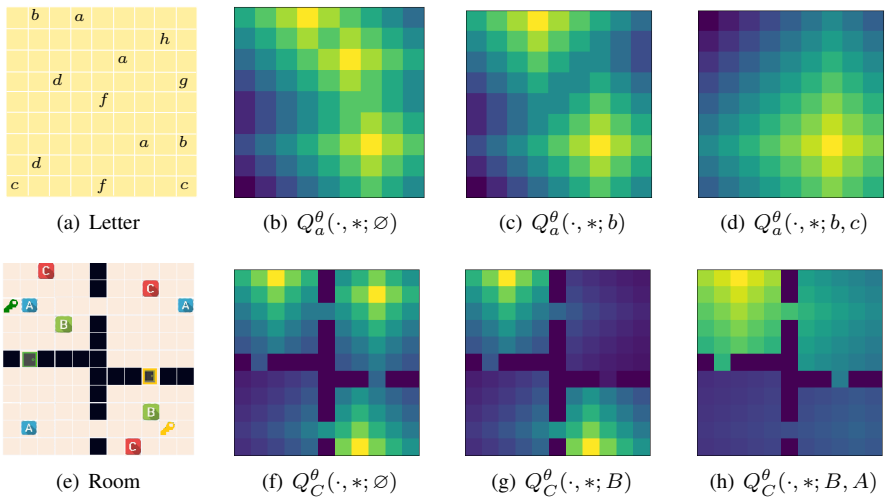


(a) $n = 9, d = 10$ (b) $n = 9, d = 12$ (c) $n = 9, d = 15$

(d) $n = 11, d = 10$ (e) $n = 11, d = 12$ (f) $n = 11, d = 15$

(g) $n = 15, d = 10$ (h) $n = 15, d = 12$ (i) $n = 15, d = 15$

Figure 8: Performance comparison for long-horizon tasks in letter domain. The x-axis is the number of environment steps taken for option training. The y-axis is the average sum of rewards received in the trajectory. The map size is $n \times n$ and the task formula has depth of d. The evaluation takes place at the steps of $\{3, 5, 10\} \times 10^6$, during the option training.



(a) Letter (b) $Q_a^\theta(\cdot, *; \varnothing)$ (c) $Q_a^\theta(\cdot, *; b)$ (d) $Q_a^\theta(\cdot, *; b, c)$

(e) Room (f) $Q_C^\theta(\cdot, *; \varnothing)$ (g) $Q_C^\theta(\cdot, *; B)$ (h) $Q_C^\theta(\cdot, *; B, A)$

Figure 9: Visualization of trained action-value function of options. The first row is for the option of reaching a in letter domain, and the second row is for the option of reaching C in room domain. The color in every grid (state s) corresponds to the Q value of the optimal action, i.e., $\forall s, Q_p^\theta(s, *; \xi) = max_a Q_p^\theta(s, a; \xi)$.

The comparison results in terms of episodic return are shown in Figure 8. Since the evaluation results of long-horizon tasks have large variances, we only show the results as charts here. The TL specification for evaluation is a DNF task consisting of 3 conjunctions with the depth of d, where every letter is randomly generated without repetition. Every task here has longer horizon than that in Figure 4. Every result in Figure 8 is the average of 10 formulas, and the variance is obtained from 5 seeds. We can see that the proposed method can significantly outperform the Baseline GCN-LTL. Moreover, the superiority of our proposed method improves with map size and formula depth, showing that the proposed method can solve the long-horizon tasks well and the effect of reward propagation is significant.

A.7 Visualization

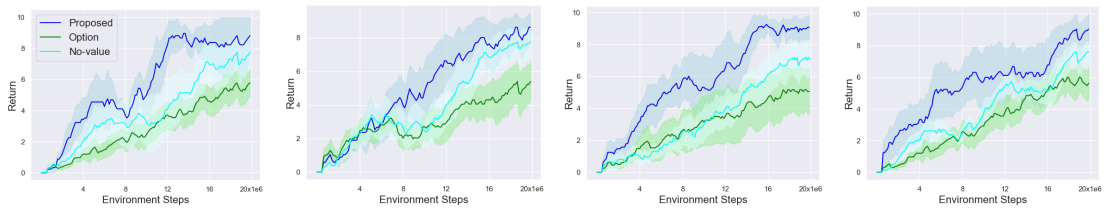
Finally, in order to show the effects of the dependence of options on future subgoals, we visualize the Q functions of the same option dependent on different future subgoals. The color of very grid represents the discounted return to the target subgoal, where the brighter the color is, the higher the return will be.

In Figure 9, the first row shows the Q functions of reaching subgoal a in letter domain in three scenarios, namely dependent on nothing, b and $b \to c$. Every grid represents the environment state where the agent is in that grid. On the map shown in Figure 9(a), there are three letters a. According to Figure 9(b), the agent should go to the closest a. Figures 9(c) and 9(d) tell us that when dependent on b or $b \to c$, the option of reaching a regards a in first row or 7-th row as the target.

In the second row of Figure 9, we can see that in room domain, the option of reaching C has different targets when the future subgoal sequences ξ are different. Specifically, In Figure 9(h), the grid containing the yellow key has the highest value in the bottom rooms and the grid having C in the upper left room has the highest value across the whole map. This indicates that in environment states where the agent is in the bottom rooms, the agent should first go to pick up the yellow key as an intermediate target and then go to C in the upper left corner. It shows that the agent successfully learns the skill of opening a lock by the right key, without having any key proposition or prior knowledge.

A.8 Additional Results in Navigation Environment

In addition to Section 4.3.1, we conduct more experiments in Navigation environment with more complex agent of car. This agent simulates a wheeled robot with differential drive control. The performance evaluation is shown in Figure 10. The setups of Nav 1 and 2 are same as that introduced in Section A.5. Here, we



(a) Nav1, DNF Task (b) Nav1, Rec Task (c) Nav2, DNF Task (d) Nav2, Rec Task

Figure 10: Performance evaluation in Navigation environment with Car agent. Nav1 and 2 are different setups of navigation environment, introduced in Section A.5.

compare the proposed framework with *Option* baseline which learns options of reaching different subgoals myopically without looking into the future, similar as that in (León et al., 2020; 2021). Then the proposed framework without multi-step value function is also evaluated, denoted as *No-value*. We can see that, in all these experiments, the option baseline performs the worst, showing the importance of future subgoals. The no-value baseline can reach similar performance as the proposed framework, but it has a much slower convergence speed, demonstrating the effect of multi-step value function on accelerating the value propagation and learning speed.

A.9 Neural Network Architecture

The agent's architecture of critic (Q or V function) is shown in Figure 11. The input consists of observation, subgoal embedding and subgoal sequence. The observation is processed by the perception module. The subgoal embedding is the one-hot encoding of the subgoals in $\mathcal{G}$. The future subgoal sequence is processed by GNN or GRU. After inputs are processed, the embeddings of observation, subgoal and future subgoal sequence are concatenated and fed into an MLP to predict the return. The multi-step value function V^{ϕ} has the same architecture as the critic function, except that its inputs are only the observation and future subgoal sequence.

The perception module is determined by the observation space of the environment. In letter/room domain with map size of $n \times n$, we used a 3-layer convolutional neural network (CNN) which have 16, 32 and 64 channels, respectively, where the kernel size is $l \in \{2, 3, 4\}$ and stride is 1. In navigation domain, we used a 2-layer fully-connected network with [256, 256] units and ReLU activations.

The sequence of future subgoal is processed by GNN or GRU here. The GNN used here is a graph convolutional network (GCN) (Kipf & Welling, 2016; Schlichtkrull et al., 2018) with 8 message passing steps and 32-dimensional node embeddings. The GRU used here is a 2-layer bidirectional GRU with a 32-dimensional hidden layer.

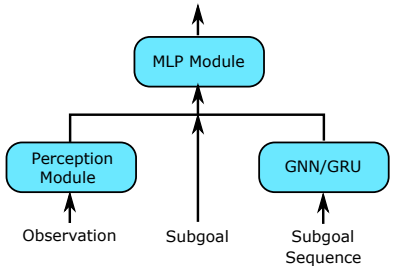


Figure 11: Neural Architecture of $Q_p^{\theta}(\cdot, \cdot; \xi)$ or $V_p^{\theta}(\cdot; \xi)$, where p is the subgoal and ξ is the future subgoal sequence.

For the MLP part of the critic function in Figure 11, we use 3 fully-connected layers with [64, 64, d_a] units and ReLU activations for all three domains. For discrete action space environments, a_d is the number of possible actions, and the output of critic function was passed through a logit layer before softmax. For the continuous case, a_d is the action dimension and we also need to train an actor network sharing same architecture as the critic network except the Tanh activation. Then we assume a Gaussian action distribution and parameterized its mean and standard deviation by sending the actor's output to two separate linear layers.

In three baselines, the Q/value networks and actor network of the agent have the same architectures introduced here, keeping the same model complexity as the proposed method. In baseline Option, since the option does not consider future subgoals, the critic network does not have any module to process the subgoal sequence. In baselines

GCN-LTL and GRU-LTL, since they do not use options, the critic network and actor network do not have any subgoal as its input, where TL specification is first transformed into a syntax tree and processed by a GCN (in baseline GCN-LTL) or GRU (in baseline GRU-LTL without progression). The GCN has the same architecture as that introduced above. The GRU in baseline-3 is also a 2-layer bidirectional GRU with 32-dimensional hidden layers.

A.10 Practical Implementation Techniques

Training Curriculum. In the option training, the agent is trained to satisfy a randomly generated subgoal sequence ξ with maximal environment return. Denote the maximum length of ξ as K. The training curriculum consists of K levels. As such, in the k-th level ($k = 1, \ldots, K$), the length of subgoal sequence ξ is set to be k. Whenever the average success rate in k-th level is above a threshold (e.g., 80%), the agent will proceed to $(k+1)$-th level where the length of subgoal sequences becomes longer. Therefore, the difficulty of tasks increases gradually as the agent proceeds to higher levels. For any subgoal sequence ξ, the agent applies options to satisfy subgoals in ξ one-by-one with conditions of future subgoals. The details are introduced in Algorithm 1. In letter and room domains, we use deep Q learning (Mnih et al., 2015) (off-policy) to train options, whereas in the navigation domain, we use PPO (Schulman et al., 2017) algorithm (on-policy) to train options. The details of hyper-parameters are introduced in Section A.12.

Table 1: Hyperparameters of PPO in Navigation Domain

Hyperparameter	Value
Env. steps per update	2048
Minibatch size	256
Discount	0.995
Time horizon of an episode	1000
Total number of steps	15e6
Satisfaction Reward R_F	10
HER trajectory modification ratio	1.0
Evaluation interval (episodes)	100
Evaluation episodes	10
Optimizer	Adam
Learning rate	3×10^{-4}
GAE-λ	0.95
Entropy coefficient	0.01
Value loss coefficient	0.5
Gradient clipping	0.5
PPO clipping (ϵ)	0.2

Adversarial Scheme. We also adopt an adversarial scheme for selecting training options which can improve the learning efficiency in empirical experiments. In the k-th level, at the beginning of each episode with initial state s_0, multiple subgoal sequences with the same length are randomly generated, i.e., $\{\xi_i\}_{i=1}^{N_T}$, and the j-th sequence with the lowest value is selected as the training task for the agent, i.e., $j = \arg\min_{i=1,\ldots,N_T} V^{\phi}(s_0; \xi_i)$. This implies that a difficult task in the current level is selected to train the agent, always pushing forward the capability of the learning agent.

Hindsight Experience Replay In early learning stage, most trajectories produced by agent's policies cannot achieve or satisfy the given task, which cannot provide any useful reward information to train agent's policy and value functions. Therefore, in training the options, in order to improve the learning efficiency, we propose to modify the hindsight experience replay (HER) (Andrychowicz et al., 2017) to better utilize the past unsuccessful trajectories. We extend HER to temporal logic domain by modifying any unsuccessful trajectory whose given task was not successfully finished. Specifically, in any unsuccessful trajectory τ associated with subgoal sequence ξ (ξ is not finished by τ), we find ξ' which is the subgoal sequence satisfied by τ actually and replace ξ by ξ', so that the trajectory τ associated with ξ' becomes a successful trajectory (ξ' is satisfied by τ). Then, assigning a large positive reward R_F at the time step when $\xi'[-1]$ becomes satisfied, designating the trajectory τ successful and hence useful to the training.

A.11 Algorithms

We summarize the detailed operations in option training and task execution (evaluation) in Algorithms 1 and 2, respectively. Algorithm 1 trains the Q function in line 21 via off-policy method, which can be trivially extended to train V function via on-policy method. The algorithm of extracting subgoal sequences from the TL task is presented in Algorithm 3.

Algorithm 1 Option Training Algorithm

Environment MDP $\mathcal{M}_e$; labeling function L; positive reward for task completion R_F; The set of propositions $\mathcal{P}$ and subgoals $\mathcal{G}$; multi-step value function $V^{\phi}(s;\xi)$; Q function of option policy $Q_g^{\theta}(s,a;\xi)$ for $\forall g \in \mathcal{G}$; the subgoal planner Γ^{ϑ}; replay buffer $\mathcal{B}$; trajectory buffer $\mathcal{B}_t$; episodic buffer $\mathcal{E}$; maximum length of subgoal sequence K; performance threshold ζ of upgrading to next level
Initialize parameters θ, ϑ and ϕ;
Initialize $\mathcal{B} \leftarrow []$;
% levels from 1 to K;
for $k = 1, \ldots, K$ **do**
    % train the options;
    while the average success rate is below ζ **do**
        Initialize $\mathcal{E} \leftarrow []$;
        Reset environment $s \leftarrow s_0$;
        Randomly generate N_S subgoal sequences, and select ξ with lowest value on V^{ϕ};
        for $l = 1, \ldots,$ len(ξ) **do** # *len*(ξ) *denotes the length of* ξ
            $\tilde{s}_0 \leftarrow s$;
            for $t = 0, \ldots, T_S - 1$ **do**
                Apply option policy $\pi_{\xi[0]}^{\xi[1:]}$ into the environment $\mathcal{M}_e$;
                Obtain reward r_t and next state $\tilde{s}_{t+1}$;
                Store experience tuple $(\tilde{s}_t, a_t, r_t, \tilde{s}_{t+1}, \xi[0], \xi[1:])$ into $\mathcal{E}$ and $\mathcal{B}$;
                if $L(\tilde{s}_{t+1}) \models \xi[0]$ **then**
                    Set $s \leftarrow \tilde{s}_{t+1}$ and $\xi \leftarrow \xi[1:]$;
                    Go to 10;
                end if
                Sample a minibatch $\mathcal{B}_M$ from $\mathcal{B}$ and update Q function according to (5);
                Sample trajectories from $\mathcal{B}_t$ and update V^{ϕ} according to (3);
            end for
            Break; # *the trajectory* $\mathcal{E}$ *is unsuccessful and needs to be relabeled*
        end for
        if $\mathcal{E}$ is unsuccessful **then** # *relabel unsuccessful trajectory by HER*
            Randomly select subgoal sequence ξ' satisfied by $\mathcal{E}$;
            Relabel the subgoal and condition (future subgoal) of every tuple in $\mathcal{E}$ based on ξ';
        end if
        Store transitions of $\mathcal{E}$ into $\mathcal{B}$;
        Store $\mathcal{E}$ into $\mathcal{B}_t$;
    end while
end for

A.12 Hyper-parameters

In the proposed framework, we use deep Q learning (Mnih et al., 2015) to learn options in letter and room domains, while we use PPO (Schulman et al., 2017) in the navigation domain. All experiments were conducted on a compute cluster using 1 GPU (RTX 2080 Ti). The hyper-parameters used for deep Q learning in letter and room domain are introduced in Table 2. The hyper-parameters for PPO in navigation domain are presented in Table 1. The agents in three baselines are trained by the same RL algorithms in the proposed method, using the same algorithm hyperparameters of the proposed method. In baseline Option, we do not consider any future subgoal sequence. In baselines GCN-LTL and GRU-LTL, we cannot use TTL transformation or HER since the TL specification is transformed into a syntax tree from its original form.

Algorithm 2 Task Execution Algorithm

The environment $\mathcal{M}_e$; labeling function L; the set of propositions $\mathcal{P}$; progression function $\text{prog}(\cdot,\cdot)$ introduced in (León et al., 2021); multi-step value function V^ϕ and critics of options Q_g^θ for $\forall g \in \mathcal{G}$ trained by Algorithm 1; the threshold of closeness κ; the test task specification φ;
Reset environment and obtain the initial state s_0;
Transform task specification φ into a set $\mathcal{K} = \{\xi_i\}_{i=1}^{M_\varphi}$ of accepting subgoal sequences by using Algorithm 3;
Given φ, obtain the set of unsafe subgoals $\mathcal{U}_s$;
Select ξ^* with largest value such that $\xi^* = \arg\max_{\xi \in \mathcal{K}} V^\phi(s_0; \xi)$;
set $t \leftarrow 0$;
while every sequence $\xi \in \mathcal{K}$ is not empty **do**
    Sample action a_t from the option policy $\pi_{\xi^*[0]}^{\xi^*[1:]}(\cdot|s_t)$ until $\forall g \in \mathcal{U}_s, Q_g^\theta(s_t, a_t; \varnothing) < \kappa$
    Obtain next state s_{t+1};
    if $L(s_{t+1}) \models \xi^*[0]$ **then**
        Progress the formula $\varphi \leftarrow \text{prog}(L(s_{t+1}), \varphi)$
        Update the set $\mathcal{U}_s \leftarrow \{q | q \in \mathcal{P}_G, \text{prog}(q, \varphi) = \text{false}\}$;
        $\forall \xi \in \mathcal{K}$, if $L(s_{t+1}) \models \xi[0]$, then ξ.pop($\xi[0]$)
        Select again $\xi^* = \arg\max_{\xi \in \mathcal{K}} V^\phi(s_{t+1}; \xi)$;
    end if
    $t \leftarrow t + 1$
end while

Algorithm 3 Transforming task specification into a list of subgoal sequences (León et al., 2021)

Task specification φ; the set of propositions $\mathcal{P}$;
Initialize $\mathcal{K} \leftarrow \{\}$;
for each atomic task $p \in \varphi$ **do**
    if p is atomic positive or negative **then**
        for all Seq $\in \mathcal{K}$ **do**
            Seq.append(p)
        end for
    else
        # There are non-determinstic choices
        Initialize choice list: CL
        LK $\longleftarrow$ len($\mathcal{K}$)
        for all Seq $\in \mathcal{K}$ **do**
            for all choice $\in p$ **do**
                CL.append(choice)
                Generate a clone per choice Seq$'$ $\longleftarrow$ Seq
                $\mathcal{K}$.append(Seq')
            end for
        end for
        Initialize counter $c \longleftarrow -1$
        for $i = 1, 2, \ldots,$ len($\mathcal{K}$) **do**
            if i%LK $== 0$ **then**
                $c+ = 1$
            end if
            We append a different choice to each sequence cloned $\mathcal{K}[i]$.append(CL[c])
        end for
    end if
end for
Return $\mathcal{K}$

Table 2: Hyperparameters of Deep Q Learning in Letter and Room Domain

Hyperparameter	**Value in Letter**	**Value in Room**
Batch size for training options	256	256
Batch size for training planner	64	64
Discount	0.99	0.99
Exploration ϵ init value	0.75	0.75
Exploration ϵ final value	0.05	0.05
Exploration ϵ factor	0.5	0.5
Curriculum level K	5	5
Total number of steps	10e6	10e6
Satisfaction Reward R_F	1	1
Q update interval	10	5
Q target update interval	2000	1500
V update interval	10	5
V target update interval	2000	1500
HER trajectory modification ratio	0.5	1.0
Evaluation interval	10	10
Evaluation episodes	10	10
Optimizer	Adam	Adam
Adam ϵ	2×10^{-5}	2×10^{-5}
β_1, β_2	$0.9, 0.999$	$0.9, 0.999$
Learning rate	3×10^{-4}	2×10^{-4}
Replay buffer size $\lvert\mathcal{B}\rvert$	1e6	1e6